In [1]:

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
import numpy as np
import operator
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf

sns.set_style("whitegrid")
current_palette = sns.color_palette('colorblind')
```

```
Using TensorFlow backend.
```

In [2]:

```
features = 20 #entspricht der Anzahl der Sensoren
timesteps = 22 # *0.05s --> definiert die Zeitspanne in der zeitliche Abhängigkeiten vom Netzwerk er
num_classes = 3 #Labelanzahl des Outputlayers
batchsize = 128
LSTM_size = 16 #ANzahl der LSTM-Zellen
epochen = 5

name = 'NN1_1_3Label'
```

Aufbau Model

In [3]:

```
model = Sequential()
model.add(LSTM(LSTM_size, dropout=0.3, recurrent_dropout=0.3 ,return_sequences=True,
               batch_input_shape=(None, timesteps, features)))
#model.add(LSTM(LSTM_size, dropout=0.3, recurrent_dropout=0.3 ,return_sequences=True,))
#model.add(Dense(64))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', #wird verwendet bei ont-hot-codierungen (p118 DL mit
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Trainingsdaten laden

In [4]:

```
x_train = np.load('Klassifizierung_Daten/3Label/x_train.npy').astype('float32')
x_val = np.load('Klassifizierung_Daten/3Label/x_val.npy').astype('float32')
x_test = np.load('Klassifizierung_Daten/3Label/x_test.npy').astype('float32')

y_train = np.load('Klassifizierung_Daten/3Label/y_k_train.npy').astype('float32')
y_val = np.load('Klassifizierung_Daten/3Label/y_k_val.npy').astype('float32')
y_test = np.load('Klassifizierung_Daten/3Label/y_k_test.npy').astype('float32')
```

Model trainieren

In [5]:

```
model.fit(x_train, y_train,
          batch_size=batchsize , epochs=epochen, validation_data=(x_val, y_val))
```

```
Train on 5120 samples, validate on 1152 samples
Epoch 1/5
5120/5120 [==============================] - 2s 355us/step - loss: 0.8901 - acc: 0.6188 -
val_loss: 0.6730 - val_acc: 0.8191
Epoch 2/5
5120/5120 [==============================] - 1s 158us/step - loss: 0.4970 - acc: 0.8992 -
val_loss: 0.4846 - val_acc: 0.8405
Epoch 3/5
5120/5120 [==============================] - 1s 159us/step - loss: 0.2935 - acc: 0.9382 -
val_loss: 0.4292 - val_acc: 0.8585
Epoch 4/5
5120/5120 [==============================] - 1s 160us/step - loss: 0.1923 - acc: 0.9544 -
val_loss: 0.4195 - val_acc: 0.8695
Epoch 5/5
5120/5120 [==============================] - 1s 166us/step - loss: 0.1442 - acc: 0.9618 -
val_loss: 0.4343 - val_acc: 0.8758
```

Out [5]:

```
<keras.callbacks.History at 0x1a3216c208>
```

In [6]:

```
history_dict = model.history.history
history_dict
```

Out [6]:

```
{'val_loss': [0.6729994482464261,
  0.48464330699708724,
  0.42924460603131187,
  0.4194948400060336,
  0.4342574675877889],
 'val_acc': [0.8191287848684523,
  0.8404750691519843,
  0.8585069444444444,
  0.8694760070906745,
  0.8758285906579759],
 'loss': [0.8900572150945664,
  0.49697363153100016,
  0.2934570647776127,
  0.19225387834012508,
  0.14415914379060268],
 'acc': [0.6188476555049419,
  0.8992187514901161,
  0.9382102265954018,
  0.9544122874736786,
  0.9617720156908035]}
```

Analysiere Trainingsergebnisse

In [7]:

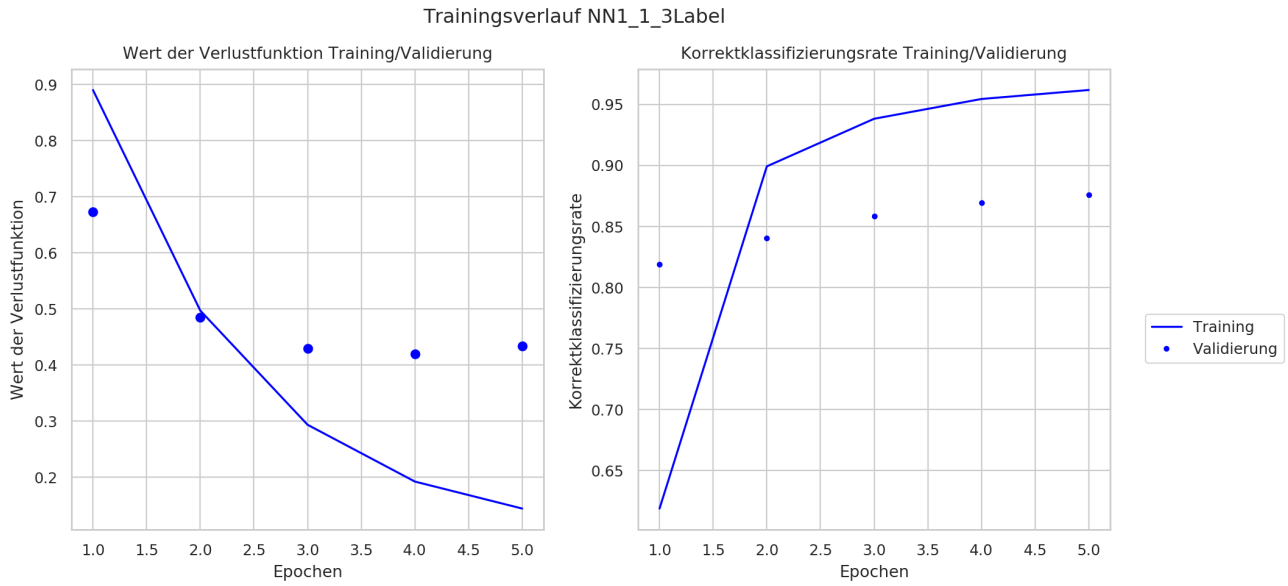
```
sns.set_context("paper")

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values)+1)

f, (ax1,ax2) = plt.subplots(1,2, figsize=(11, 5), dpi=200, facecolor='w', edgecolor='k')
f.suptitle('Trainingsverlauf '+name)
ax1.plot(epochs, loss_values, 'b')
ax1.plot(epochs, val_loss_values, 'bo')
ax1.set_title('Wert der Verlustfunktion Training/Validierung')
ax1.set(xlabel='Epochen', ylabel='Wert der Verlustfunktion')
#ax1.set_xlim(left=1, right=epochen)

acc = history_dict['acc']
val_acc = history_dict['val_acc']
epochs = range(1, len(loss_values)+1)

ax2.plot(epochs, acc, 'b', label='Training')
ax2.plot(epochs, val_acc, 'b.', label='Validierung')
ax2.set(title='Korrektklassifizierungsrate Training/Validierung',xlabel='Epochen',ylabel='Korrektkla
ax2.legend(bbox_to_anchor=(0.9, 0., 0.5, 0.5), borderaxespad=1)
f.subplots_adjust(wspace=0.2)
plt.show()
```



Anwendung des trainierten Models auf 'unbekannte' Trainingsdaten

In [8]:

```
predictions = model.predict(x_test,batch_size=batchsize)
y_real0 = y_test
```

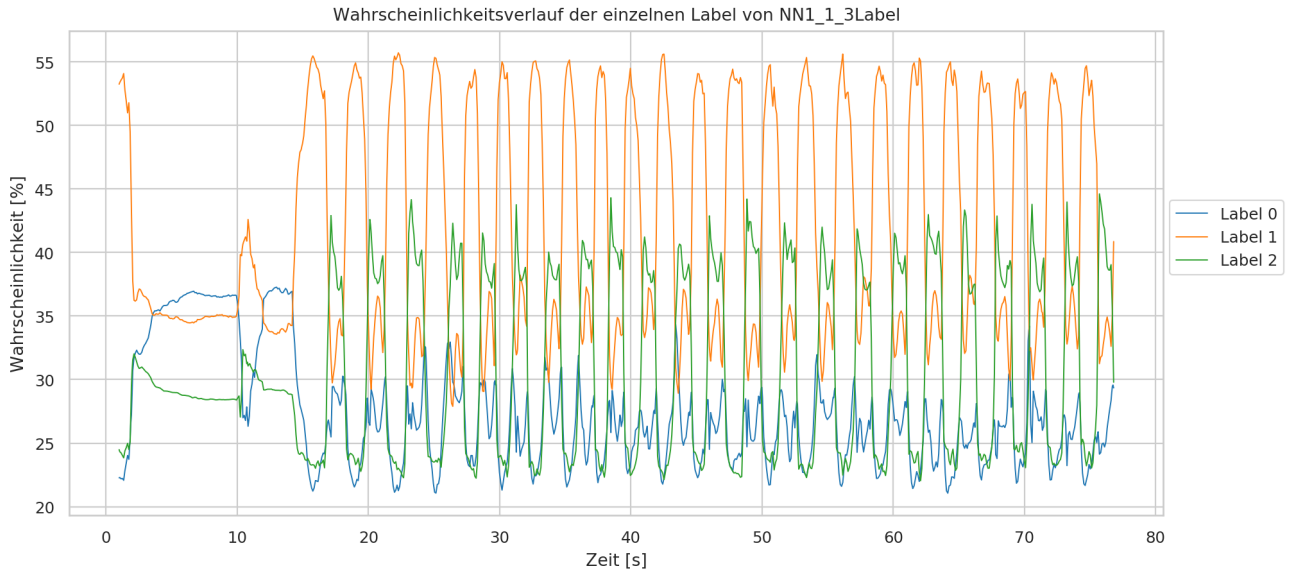
In [9]:

```
a=[[0,0,0]]
c=[[0,0,0]]
for i in range(0,predictions.shape[0]):
    a[0] = predictions[i,0]
    c = np.concatenate((c,a), axis=0)
c = c[1:]

test_out = np.zeros(len(c))
for i in range(0,len(c)):
    test_out[i] = np.argmax(c[i])
```

In [10]:

```
plt.figure(num=None, figsize=(11,5), dpi=200, facecolor='w', edgecolor='k')
plt.plot(np.linspace(1,len(c)*0.05,len(c)), c[:,0]*100, label='Label 0', linewidth=0.7)
plt.plot(np.linspace(1,len(c)*0.05,len(c)), c[:,1]*100, label='Label 1', linewidth=0.7)
plt.plot(np.linspace(1,len(c)*0.05,len(c)), c[:,2]*100, label='Label 2', linewidth=0.7)
#plt.plot(np.linspace(1,len(c)*0.05,len(c)), c[:,3]*100, label='Label 3', linewidth=0.7)
#plt.yticks([0,50,100])
plt.title('Wahrscheinlichkeitsverlauf der einzelnen Label von '+name)
plt.xlabel('Zeit [s]')
plt.ylabel('Wahrscheinlichkeit [%]')
plt.legend(bbox_to_anchor=(0.61, 0.15, 0.5, 0.5), borderaxespad=0)
plt.show()
```



In [11]:

```
a=[[0,0,0]]
b=[[0,0,0]]
for i in range(predictions.shape[0]):
    a[0] = y_real0[i,0]
    b = np.concatenate((b,a), axis=0)
b = b[1:]
```

```
y_real = np.zeros(len(b))
for i in range(len(b)):
    y_real[i] = np.argmax(b[i])
```

In [12]:

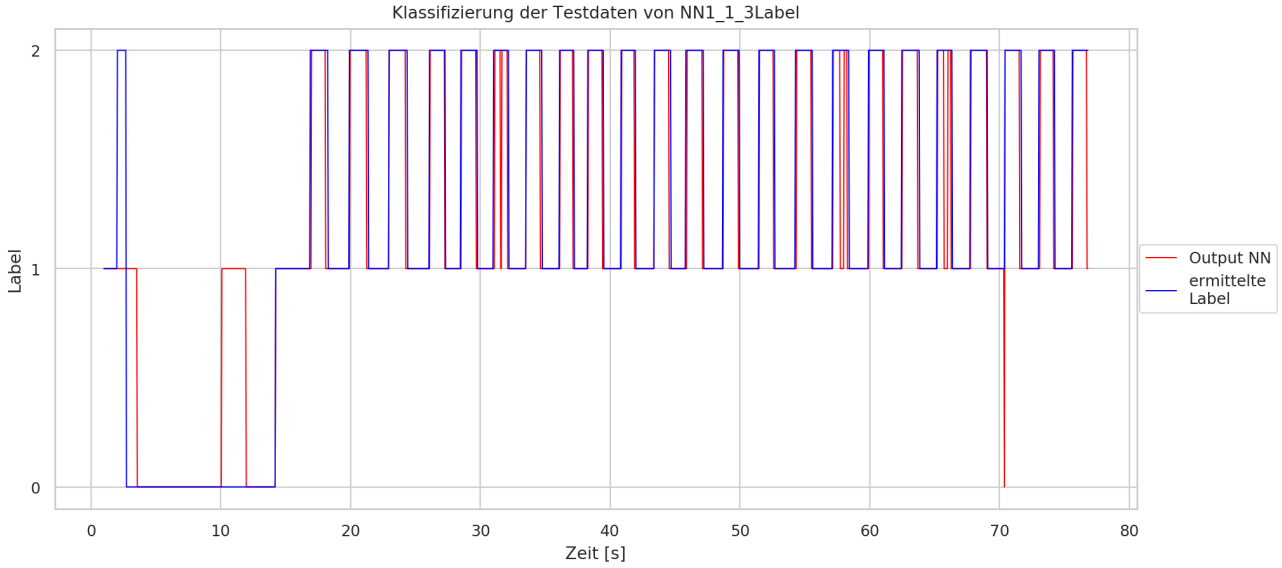
```
plt.figure(num=None, figsize=(11, 5), dpi=200, facecolor='w', edgecolor='k'),

plt.yticks([0,1,2,3])

plt.plot(np.linspace(1,len(test_out)*0.05,len(test_out)), test_out, 'r', label='Output NN',linewidth
plt.plot(np.linspace(1,len(y_real)*0.05,len(y_real)), y_real, 'b', label='ermittelte \nLabel', linew
plt.title('Klassifizierung der Testdaten von '+name)
plt.xlabel('Zeit [s]')
plt.ylabel('Label')

plt.legend(bbox_to_anchor=(0.61, 0.15, 0.52, 0.4), borderaxespad=0)
#plt.xlim(left=600, right=800)
#sns.despine(right=False)

plt.show()
```



Zusammenfassung

In [13]:

```
# Nach welcher Epoche sollte das Training optimalerweise abgeschlossen werden?
# Vorgehen: einmal Netzwerk berechnen in 25 Epochen -> optimale Epochenanzahl anhand 'min_index_val_
#           danach Netzwerk neu berechnen lassen.
min_index_val_loss, min_value_val_loss = min(enumerate(history_dict['val_loss']), key=operator.itemg
max_index_val_acc, max_value_val_acc = max(enumerate(history_dict['val_acc']), key=operator.itemgett
print('Ergebnisse der Validierungsdaten:')
print('  opimale Epochenanzahl:                                    '+str(min_index_val_loss+1))
```

```
print('  minimaler Verlust:                                          '+str(min_value_val_loss))
print('  maximale Korrektklassifizierung der Validierungsdaten:   '+str(max_value_val_acc)+'\n')

if (max_index_val_acc != min_index_val_loss):
    print('Index optimalen Verlusts und optimaler Korrektklassifizierung nicht idetisch !!! max_inde

print('Ergebnisse der Trainingdaten zur optimalen Epochenzahl:')
print('  Verlust:                    '+str(history_dict['loss'][min_index_val_loss]))
print('  Korrektklassifizierung:     '+str(history_dict['acc'][min_index_val_loss])+'\n\n')

gleiche_werte = np.sum(np.equal(test_out,y_real))
alle_testwerte = len(test_out)
print('Ergebnisse der Testdaten des trainierten neuronalen Netzes:')
print('  Anteil der Übereinstimmenden Werte: '+ str(gleiche_werte/alle_testwerte))
print('  Korrelationskoeffizient:            '+ str(np.corrcoef(test_out,y_real)[0,1]))
```

```
Ergebnisse der Validierungsdaten:
  opimale Epochenanzahl:                                    4
  minimaler Verlust:                                        0.4194948400060336
  maximale Korrektklassifizierung der Validierungsdaten:    0.8758285906579759

Index optimalen Verlusts und optimaler Korrektklassifizierung nicht idetisch !!!
max_index_val_acc bei Epoche 5


Ergebnisse der Trainingdaten zur optimalen Epochenzahl:
  Verlust:                    0.19225387834012508
  Korrektklassifizierung:     0.9544122874736786


Ergebnisse der Testdaten des trainierten neuronalen Netzes:
  Anteil der Übereinstimmenden Werte: 0.8951822916666666
  Korrelationskoeffizient:            0.88770865415724
```

In [14]:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 22, 16)            2368
_________________________________________________________________
dense_1 (Dense)              (None, 22, 3)             51
=================================================================
Total params: 2,419
Trainable params: 2,419
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
#model.save('model/'+name)
```